Neural Net-Based Continuous Phase Modulation Receivers

Gustavo de Veciana, *Student Member, IEEE,* and Avideh Zakhor, *Member, IEEE*

Abstract— **Continuous phase modulation (CPM) schemes are generally used in peak-power limited transmission systems such as digital satellite communications. Their major drawback, however, is a prohibitively complex receiver structure, particularly in modulation schemes with high packing densities. In this paper we propose feed-forward neural nets (NN) as receivers for partial response CPM systems. Our approach is to replace the entire receiver structure, excluding timing recovery, with a neural net unit whose inputs are time samples of the incoming baseband signals, and whose outputs are the decoded symbols. We present simulation results for coherent and incoherent NN based receivers, and compare their performance with the optimum maximum-likelihood (ML) receiver. A performance analysis of NN-based receivers at large SNR is presented.**

I. Introduction

CONSTANT envelope continuous phase modulation (CPM) schemes are extremely important in peak power limited communication applications such as satellite transmission systems. These schemes are generally characterized by high packing densities and prohibitively complex receiver structures [1], [2]. For instance, while their packing density increases with partial response, or the overlap of the frequency pulses L, and the alphabet size M, their optimal ML receivers require a bank of matched filters whose size grows as M^L. This bank of filters is followed by a Viterbi decoder which draws heavily on computational resources. Specifically, while reducing the modulation index h improves bandwidth efficiency, the number of states in the Viterbi decoder increases with the denominator of h.

Applications of neural networks in communication systems have been proposed in a variety of contexts such as ML sequence detection, and decoding error correction codes [3], [4]. In this paper, we develop neural network based receiver structures for constant envelope CPM systems. Our motivation is to reduce the complexity of implementation by casting the demodulation task into the more general framework of a neural network classification task. In so doing, we replace the matched filter banks and the Viterbi decoder of the optimal receiver with a feed-forward net trained to demodulate the incoming baseband signal. Although the performance of this receiver is suboptimal, it is our hope that its implementation for more complex modulation schemes, which exploit the regularity of neural network architectures will become practical as VLSI analog and/or digital neural network chips become workable commodities.

The organization of the remaining part of this paper is as follows. In Section II we review optimal receivers for CPM schemes. Section III introduces the NN-based receiver and describes simulation results of its performance. Section IV contains analytical results on noise propagation in multilayer feed-forward neural networks at large SNR, and their application to our particular problem. Section V has a brief discussion of NN classifiers' complexity and performance. Finally, Section VI includes conclusions and directions for future research.

II. Optimal CPM Receivers

In this section we briefly review the optimal maximum-likelihood CPM receiver [5]. Consider a phase modulated signal

$$s(t,\tilde{\boldsymbol{a}}) = \sqrt{\frac{2E}{T}}\cos(\omega_c t + \phi(t,\tilde{\boldsymbol{a}}) + \phi_o)$$

where

$$\phi(t,\tilde{\boldsymbol{a}}) = 2\pi h \sum_{i=0}^{\infty} a_i q(t-iT).$$

$\tilde{\boldsymbol{a}}$ is the data stream with $a_i \in \{\pm 1, \pm 3, \cdots \pm M-1\}$, for even values of the alphabet size, M; $q(t)$ is the phase pulse with a corresponding frequency pulse lasting L symbol intervals, h is the modulation index, E is the energy per symbol, ϕ_o is the phase offset and ω_c is the carrier frequency. For coherent demodulation schemes it is assumed that ϕ_o has been recovered; thus without loss of generality ϕ_o is set to zero.

When the modulation index h is rational, as $h = \frac{2k}{p}$, a CPM signal can be described by a finite-state Markov process, whose state at a given symbol interval is specified by the *correlative* state vector $v_n = (a_{n-1}, a_{n-2}, \cdots a_{n-L+1})$ determined by the last $L-1$ letters and the *phase* state $\theta_n = \left[\pi h \sum_{i=-\infty}^{n-L} a_i\right] \bmod 2\pi$. The phase state encapsulates the phase history prior to the last L letters for the nth symbol a_n. There are M^{L-1} correlative states and p phase states, $\theta_n = 0, \frac{2\pi}{p}, \cdots, \frac{(p-1)2\pi}{p}$, and therefore pM^{L-1} states in the entire state space.

In the ML receiver the incoming signal is multiplied by $\cos(\omega_c t)$ and $\sin(\omega_c t)$ and low-pass filtered in order to generate the in-phase and quadrature components. As shown in

Paper approved by the Editor for Transmission Systems of the IEEE Communications Society. Manuscript received October 26, 1990; revised April 19, 1991. This work was supported in part by Joint Services Electronics Project, and NSF presidential young investigator award MIP-9057466. The work of G. de Veciana was supported by NSF graduate fellowship. This paper was presented in part at the 1990 International Conference on Communications, Atlanta, GA.

The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

IEEE Log Number 9201154.

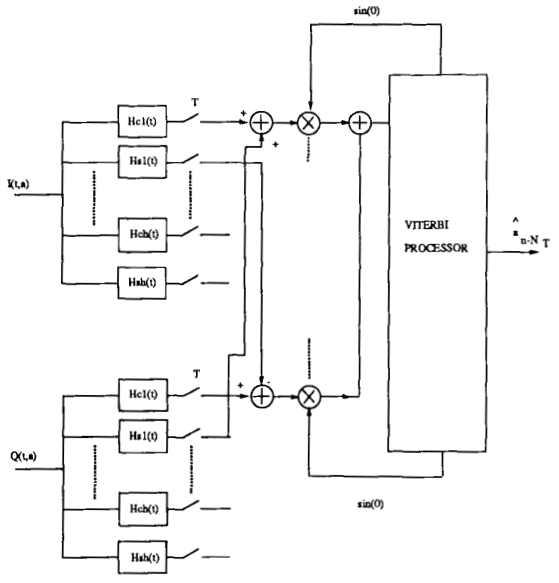

Fig. 1. Conventional optimum receiver.

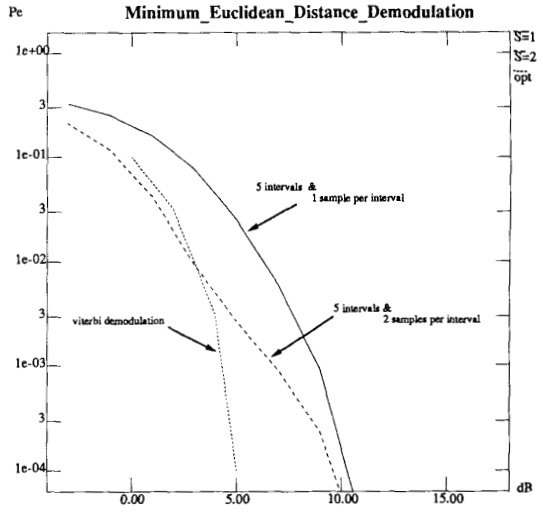

Fig. 2. Probability of error versus SNR for minimum Euclidean distance.

Fig. 1, these components are then fed into $2 \cdot M^L$ matched filters which effectively calculate the correlation between the received signal and all possible transmitted signals over one interval given the present phase state [6]. Linear combinations of the sampled outputs of the matched filters are used by the Viterbi algorithm to decode the transmitted symbols.

Incoherent demodulation is a significantly harder problem than coherent demodulation. The signal can no longer be characterized by a finite set of states, resulting in a cumbersome theoretical formulation that requires the calculation of expectation integrals with respect to the unknown signal phase [5]. The suboptimal incoherent receivers used in practice impose restrictions on the alphabet size, modulation index, and the partial response overlap [5]. In order to fully exploit the bandwidth efficiency of CPM schemes, one cannot remain within these constraints.

In the next section we present a class of receivers which could be used for both coherent and incoherent demodulation with arbitrary phase pulse, modulation index and alphabet size. Their performance is then compared with the optimal ML receiver.

III. Neural Network Receivers

In order to motivate the use of neural networks we first discuss some related issues in CPM demodulation. This is followed by a more detailed introduction in Section III-B. Simulations are presented in Section III-C.

A. Why Neural Networks?

We propose to demodulate the incoming signal on the basis of samples from a moving time window about the symbol interval of interest. This choice is analogous to sliding-block schemes recently proposed as a parallelizable algorithm for Viterbi decoding [7]. Such algorithms exploit the well known idea of finite truncation depth in calculating the distance metrics for the Viterbi algorithm. The viability of these methods for decoding convolutional codes was studied by Heller and Jacobs [8]. In their study they show that survivor paths are likely to merge 4 to 5 constraint lengths[1] into the trellis regardless of the initial state. Thus if one allows time for the survivor paths to merge (or synchronize) and then allows for an acceptable truncation depth before making decisions, one can obtain close to optimal performance [7].

The above obviates the idea of demodulating on the basis of a finite number of past and future symbol intervals. Moreover, sliding block decoders can be implemented by a brute force approach such as table lookup [9]. However table lookup quickly becomes impractical as the constraint length increases, due to the memory requirements.

For CPM the constraint length is analogous to the parameter L or partial response in the system. Accordingly, phase trellis paths should merge before some appropriate number of symbol intervals proportional to L. As above we consider a finite window of past and future intervals about the one which is to be demodulated and find the path of minimum Euclidean distance. Once again, since there are only a finite number of possible paths, it suffices to calculate the vector (path) with minimum distance from the incoming signal to determine the transmitted symbol. Fig. 2 shows the performance of such an approach versus that of Viterbi demodulation with a truncation depth of 11 intervals [5]. As seen, the performance is almost as good as Viterbi for low SNR and improves as the number of samples per interval are increased. Although the results are good, this approach is not feasible due to the complexity of evaluating the minimum distance. For instance the case of 2 samples per interval requires approximately 12 000 multiplies per decoded symbol.

The motivation for using neural networks, lies in their ability to approximate functions. Both analytical and experimental results have been reported verifying the ability

[1] The amount of memory in the convolutional coder.

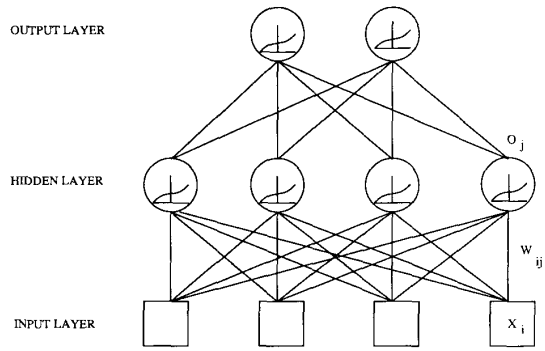

Fig. 3. Generic architecture.

of multilayer feed-forward neural networks to learn, and approximate arbitrarily complex nonlinear functions, [10], [11]. Funahashi reports a theoretical result, demonstrating the generality of two-layer feed-forward network approximation for continuous functions, using Sigmoid nonlinearities [12]. Lapedes and Farber present an intuitive geometrical approach to explaining NN operation in performing interpolation or extrapolation [13]. These two functions can alternatively be used for signal processing or symbol processing applications. For our application the desired function is the demodulation map, bypassing an explicit calculation of Euclidean distances and metric comparisons. Clearly the neural net implementation will be approximate. We conjecture, however, that the NN's "internal representation" will make use of the special structure of a modulated signal to render the demodulation map effectively [14]. The remainder of this section is dedicated to an investigation of this idea with respect to the problem of CPM demodulation, and some simulations.

B. Neural Network-Based Receiver

A feed-forward neural network has one or more layers of identical nonlinear units, which are densely interconnected from one layer to the next by variable weight links [15]. An example of a two layer network is shown in Fig. 3. The input layer branches scaled versions of the inputs to the first nonlinear layer, commonly called a *hidden layer.* Hidden nodes are those whose outputs are not directly available. In this example there is only one hidden layer feeding into the output nodes. The nonlinear hidden and output nodes compute the composition of a Sigmoid with the weighted sum of outputs from the previous layer:

$$o_j = \frac{1}{1 + \exp\left(\dfrac{-\sum_i w_{ji} x_i + \theta_j}{T}\right)} \tag{1}$$

where x_i denotes the ith input, w_{ji} denotes the weights from the ith input node to the jth hidden node, T denotes the "temperature" controlling the sharpness of the nonlinearity (see Fig. 4) and θ_j and o_j denote the bias and output of the jth hidden node, respectively.

The continuous phase demodulation problem is an inherently well defined albeit complex one, and therefore provides

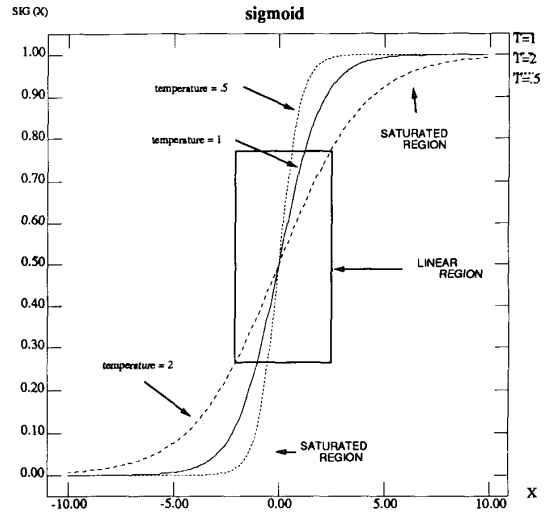

Fig. 4. Sigmoids for different temperatures.

a natural context for investigating the robustness of neural net classifiers. In applying neural networks to CPM demodulation a number of issues need to be addressed. These include the following.

1) Choice of the inputs and output of the network.
2) Network architecture, e.g., the number of layers, the number of nodes per layer, the nonlinearity function used at each node, and the connectivity between the layers.
3) Generation of the training set.
4) Training procedure.
5) Performance characterization.

In what follows we discuss each of these issues separately.

1) Representation: The input vector consists of the sampled baseband signal windowed about the symbol interval to be demodulated. The observation window spans several intervals centered about the interval of interest. Since CPM signals with partial response L spread the information associated with a given symbol over L intervals, we select an observation window of *minimum* length $O = 2L - 1$. Thus, for the nth symbol interval, the signal is sampled over $\left[\left(n - \frac{O+1}{2}\right)T, \left(n + \frac{O+1}{2} - 1\right)T\right]$. Strictly speaking, one sample per interval per quadrature component suffices to completely specify a unique phase path to within a phase offset ϕ_o. However, we found that increasing the number of samples per interval S improves the network's performance. Thus, the input is an $I = 2 \times O \times S$ dimensional vector, consisting of samples from both the in-phase and quadrature components of the received baseband signal over an observation window of length O intervals.

The output corresponds to a binary representation of the M symbols in the alphabet. The number of output nodes is thus at least $\log_2 M$. Each output node is followed by a hardlimiter which clips the output signal to either 0 or 1. For a given input vector, the output corresponds to the symbol associated with the interval about which the observation window is centered.

2) Network Architecture: The above input/output selection determines the number of input and output nodes, but leaves

the number of hidden nodes unspecified. Much effort has gone into quantifying the number of hidden nodes needed for a given problem [16], [17]. However the results are limited and do not take potentially noisy inputs into account.

3) Generating the Training Set: An element in the training set, denoted by (X, Y), consists of an input vector X and the desired output Y, with components x_i and y_i, respectively. For coherent demodulation there is a finite set of phase states resulting in a finite number of possible noiseless signals within a given observation window. The training set is formed by systematically generating all these signals, and sampling them S times per interval. The size of the complete set is $pM^{L-1} \times M^O$ where the first term corresponds to the number of possible initial states depending on θ_n and the correlative state vector, and the second term corresponds to the total number of possible transmitted symbol sequences over O intervals.

The number of exemplars (X, Y) for incoherent demodulation is no longer finite, and one must arbitrarily decide the size of the training set. In this case the input vectors are samples of curves in the hyperdimensional space $\mathcal{R}^I$, corresponding to each possible sequence of transmitted symbols, and parametrized by the unknown phase ϕ_o. There are $M^{L-1} \times M^O$ such curves to be distinguished by the network, and we choose $s =$ 5, 10, 20 evenly spaced samples from each curve, so the size of the training set is $s \times M^{L-1} \times M^O$

4) Training: Error-backpropagation was used to train the networks [14]. This is an iterative gradient descent algorithm which minimizes the error between the actual and desired outputs, by adjusting the network's weights. For each sample in the training set two steps are executed.

1) Compute the error between the actual output due to the existing weights and the desired output associated with the input vector in the training set.
2) Backpropagate the error signal from the output layer towards the input layer in order to update the weights.

The samples are arranged in random order to avoid biasing the network for any particular vector.[2] The convergence criterion for the training process is

$$\text{MSE} = \frac{1}{N} \sum_i (d_i - y_i)^2 < 0.01^2$$

where the summation is over the exemplars in the training set, and d_i and y_i are the desired and the actual output for the ith exemplar. The convergence of this procedure is not guaranteed, although some adjustments can be made in order to improve convergence. Two parameters generally referred to as the *learning rate* and *momentum,* control the extent to which the weights change from one iteration to the next. For the simulations presented in this paper the momentum and learning rate are, respectively, 0.5 and 1.

5) Performance Evaluation: Our proposed NN-based receiver operates on samples of the incoming signal in a sequential manner. It decodes the current symbol, shifts the input samples, and decodes the next symbol. The receiver's performance however, is measured by the probability of error P_e, for a large collection of noisy input vectors at a given signal-to-noise ratio (SNR). The channel is simulated by adding identically distributed independent Gaussian noise samples to the input samples. The bandwidth of the receivers front-end low-pass filter is assumed to be the -20 dB bandwidth of the CPM signal.

[2] It is not clear what an optimal ordering would be.

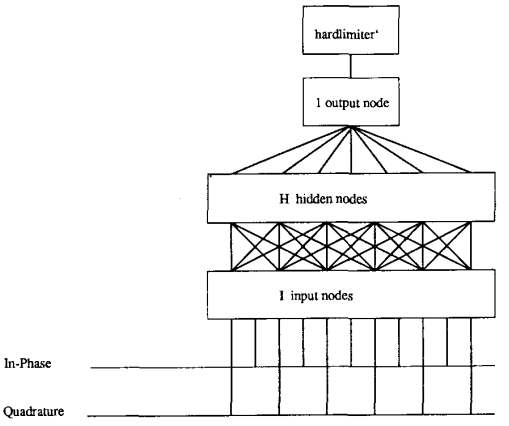

Fig. 5. Prototype of neural net receiver.

TABLE I
SUMMARY OF DIFFERENT ARCHITECTURES EXAMINED FOR COHERENT CASE

O	S	H	number of inputs	trainging set size
5	1	20	10	640
5	2	10	20	640
5	2	20	20	640
5	2	30	20	640
7	1	20	14	2560
7	2	10	28	2560
7	2	20	28	2560
7	2	30	28	2560
9	1	20	18	10240
9	2	10	36	10240
9	2	20	36	10240
9	2	30	36	10240

C. Simulations

A general purpose neural net simulator[3] was used to perform tests on different topologies and modulation schemes. We denote a scheme with a raised cosine frequency pulse by LRC where L is the partial response length or memory of the signal [5]. Tests were conducted for coherent and incoherent binary 3RC with modulation index $h = 0.8$. The probability of error was estimated by simulating the receiver until 100 errors were observed, for various SNR. Some further considerations on estimating P_e can be found in the appendix.

1) Coherent Binary 3RC: Fig. 5 shows the prototype of the networks that were considered. The variable parameters are the number of intervals in the observation window $O = 5$, 7, 9, the number of samples per interval $S = 1$, 2, and the number of hidden nodes $H = 10$, 20, 30. Table I shows the relative sizes of the input and the training sets for different choices of these parameters.

[3] Rochester Connectionist Simulator.

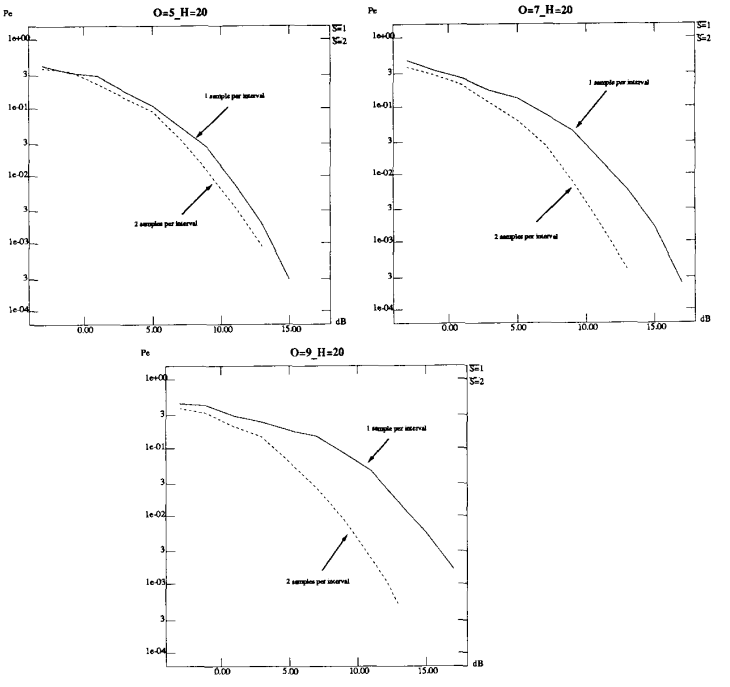

Fig. 6. Effect of increasing sampling, $H = 20$.

The sequence of plots in Fig. 6 shows P_e versus SNR for three sets of networks with different observation window length and sampling rate, but a fixed number of hidden nodes. From these plots it is clear that improved performance can be obtained with $S = 2$. Note, however, that increasing O from 5 to 7 to 9 while keeping $H = 20$, does not necessarily improve performance; this will be discussed more thoroughly in Section IV.

For the plots in Fig. 7 the number of hidden nodes H is increased for different observation intervals O, while $S = 2$. As seen, increasing the number of hidden nodes improves the receiver's performance. However, the incremental improvement decreses as the number of hidden nodes becomes too large. A good rule of thumb for achieving the best performance for a fixed number of inputs is to have approximately as many hidden nodes as input nodes. In so far as the classification of the training set vectors is concerned, many of the hidden nodes are redundant. Indeed, we found that removing hidden nodes with weak links to the output, did not hinder the network's ability to classify the training set. However these nodes do have an impact when noise is introduced.

Fig. 8 shows the superposition of the best performance curves corresponding to each plot in Fig. 7, together with that of the simulated optimal receiver with a truncation depth of $N_T = 11$ [5]. While increasing O and H results in a significant improvement at high SNR, the NN receiver performance is still $\approx$5 dB worse than the optimal at $P_e = 10^{-3}$. Possible ways to reduce this performance gap include increasing the sampling rate to exploit noise correlation, feedback of previous decoded symbols, or improving the training scheme.

2) Incoherent Binary 3RC: The incoherent NN receiver is attractive because it integrates the demodulation and phase recovery modules into one. As explained in Section III-A, the training set for incoherent demodulation is always incomplete since there are infinitely many exemplars.

Fig. 9, shows the performance of an incoherent NN receiver for binary 3RC. As expected the receiver's performance is improved by increasing the number of training vectors (p) corresponding to different possible initial phases ϕ_0, from 5 to 10 to 20. In addition, as in the coherent case, an increase in the observation interval and the number of hidden nodes improves the receiver's performance. Finally, note that the input vectors used to test the incoherent receiver have a truly random initial phase ϕ_0, which will not necessarily coincide with those in the training set.

IV. Performance Analysis

In this section we present an approximate performance analysis for a multilayer feed-forward neural net classifier at high SNR and apply the results to our NN based receiver.

1) Characterization: Consider the prototype two-layer feed-forward net shown in Fig. 5. For a given input vector in the training set we define the nominal value m_j of a node to be the linear combination of noise free inputs going into that node. If independent Gaussian noise n_i of variance σ_n^2 is added component-wise to the inputs x_i, the output of the jth hidden node can be expressed as

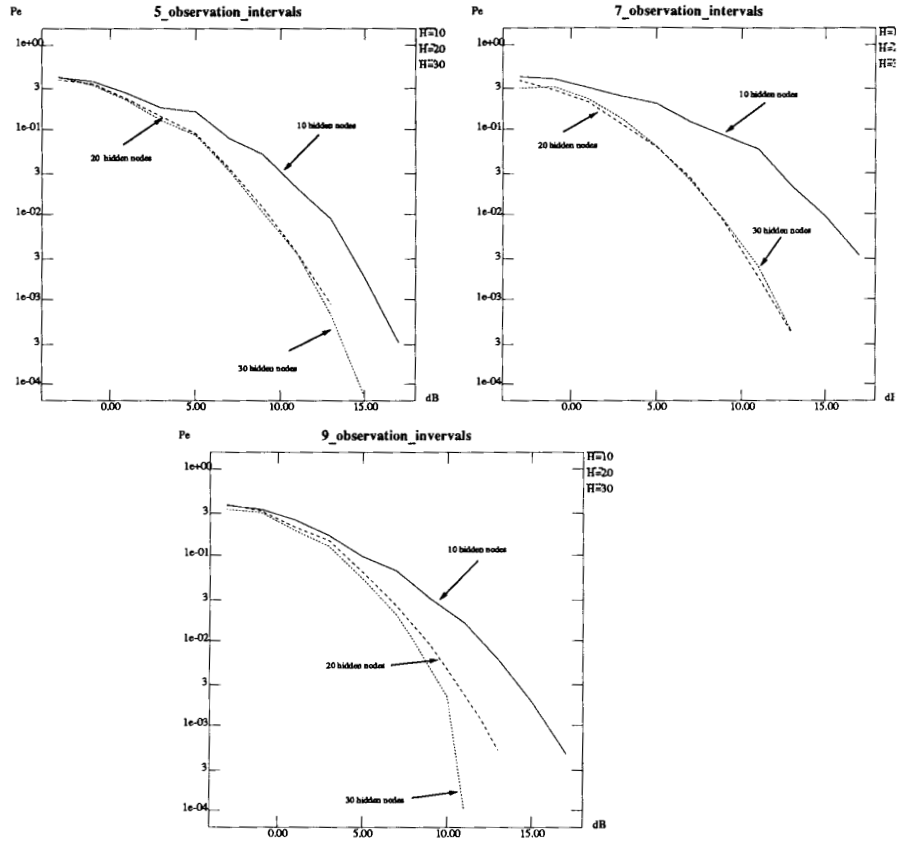

Fig. 7. Effect of increasing hidden nodes, $S = 2$.

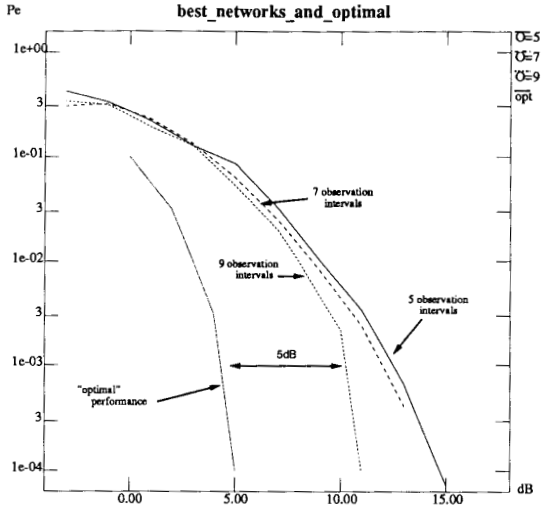

Fig. 8. Comparison of best performance plots.

$$o_j = \sigma\left(m_j + \sum_{i=1}^{I} w_{ji} n_i\right), \quad m_j = \theta_j + \sum_{i=1}^{I} w_{ji} x_i \qquad (2)$$

where $\sigma(\cdot)$ denotes the Sigmoid nonlinearity. For n_i sufficiently small (2) can be linearized about the node's nominal value to obtain,

$$o_j \approx \sigma(m_j) + \dot{\sigma}(m_j) \times \sum_{i=1}^{I} w_{ji} n_i \qquad (3)$$

where $\dot{\sigma}(\cdot)$ denotes the derivative of the Sigmoid (Fig. 10). Consider the propagation of the noise component n_i from the ith input node to the output. As seen in (3) n_i is scaled by $\dot{\sigma}(m_j) w_{ji}$ upon reaching the jth hidden node, whence it is scaled by α_j before reaching the output. By summing over all hidden nodes we obtain the total contribution of n_i to the output signal,

$$\sum_{j=1}^{H} \dot{\sigma}(m_j) \alpha_j w_{ji} \times n_i.$$

This corresponds to a noise variance of

$$\left\{\sum_{j=1}^{H} \dot{\sigma}(m_j) \alpha_j w_{ji}\right\}^2 \times \sigma_{n_i}^2.$$

Finally, by summing over all input nodes we obtain the noise

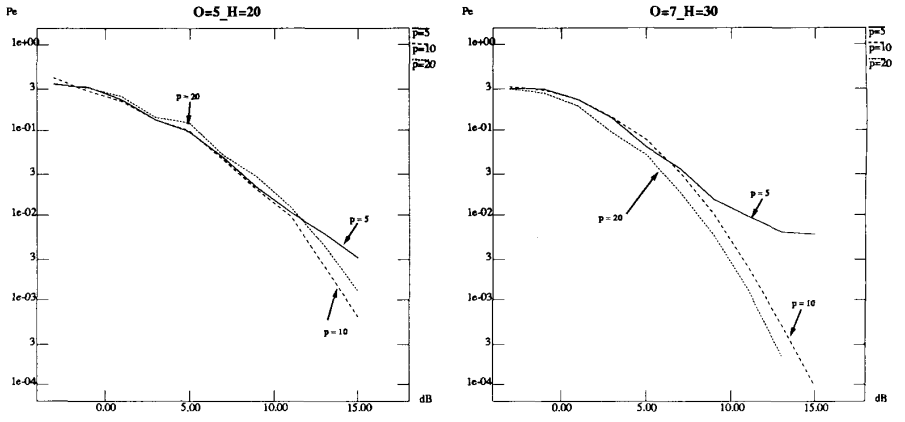

Fig. 9. Performance plots for incoherent 3RC.

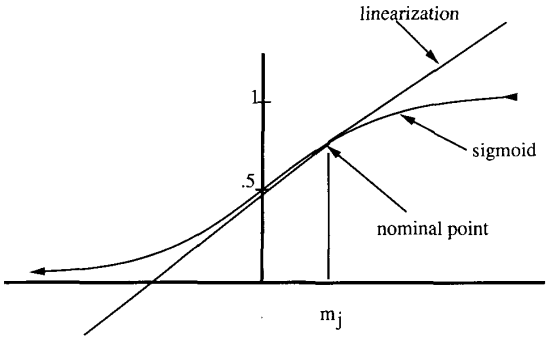

Fig. 10. Linearizing at nominal value.

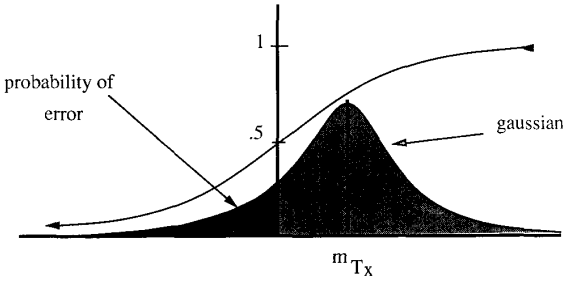

Fig. 11. Probability of error.

variance reaching the output or threshold unit for the vector X

$$\sigma_X^2 \approx \sum_{i=1}^{I} \left\{ \sum_{j=1}^{H} \dot{\sigma}(m_j)\alpha_j w_{ji} \right\}^2 \times \sigma_n^2. \tag{4}$$

If the noise variance is assumed to be small, the output signal will have a Gaussian distribution with the variance in (4). This can then be used to obtain the probability of error of the NN classifier. Later we shall see that this is not entirely justified. The ratio between the output and the input noise variance for a given vector X is defined to be its *noise factor*,

$$Nf_X \approx \frac{\sigma_X^2}{\sigma_n^2}.$$

For each input vector X the nominal value of the threshold unit, m_{T_X}, and the estimated noise variance $Nf_X\sigma_n^2$, can be used to obtain the probability of incorrectly classifying that vector,

$$P_{e_X} \approx Q\left(\frac{|m_{T_X}|}{\sqrt{Nf_X\sigma_n^2}} \right)$$

where $Q(\cdot)$ is the complimentary distribution function of the standard Gaussian (Fig. 11).

Furthermore, by averaging P_{e_X} over the entire training set, we obtain the P_e for the classifier,

$$P_e \approx \frac{1}{M} \sum_{i=1}^{M} Q\left(\frac{|m_{T_{X^i}}|}{\sqrt{Nf_{X^i}\sigma_n^2}} \right) \tag{5}$$

where M is the total number of vectors X^i in the training set.

Given the shape of the nonlinearity, one might define two possible states for each node: *linear* and *saturated.* A rough definition of these might be surmised from Fig. 4 where the Sigmoid nonlinearity in (1) is plotted for different temperatures. For any input vector X in the training set, we define a set $\mathcal{L}_X$ which contains the hidden nodes operating in the linear region. By approximating the slope of each linear node by $\frac{1}{4T}$, i.e., the slope of the Sigmoid at 0, and assuming saturated nodes do not contribute to the output noise, we obtain the following expression for the noise reaching the threshold unit:

$$\sigma_X^2 \approx \left(\frac{1}{4T} \right)^2 \sum_{i=1}^{I} \left\{ \sum_{j \in \mathcal{L}_X} \alpha_j w_{ji} \right\}^2 \sigma_n^2. \tag{6}$$

Although (6) is very approximate, it does provide insight by decoupling the factors affecting the performance. These factors are: the product of weights, the temperature and the number of saturated nodes, and from (5) the nominal value of the threshold unit. Through this characterization we can establish

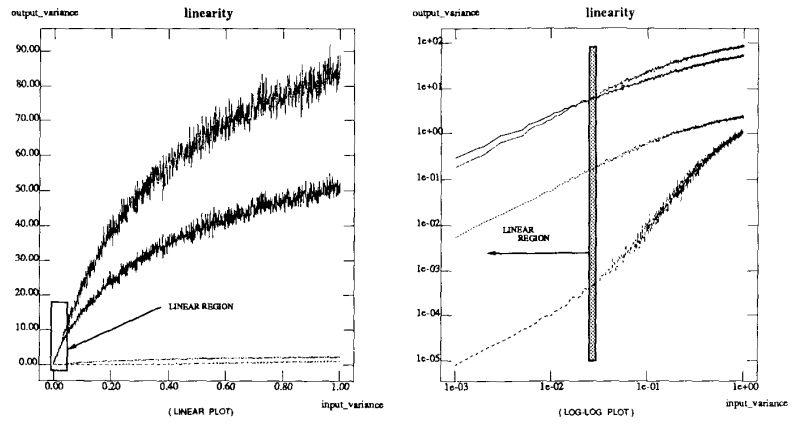

Fig. 12. Input versus output variance for a NN receiver.

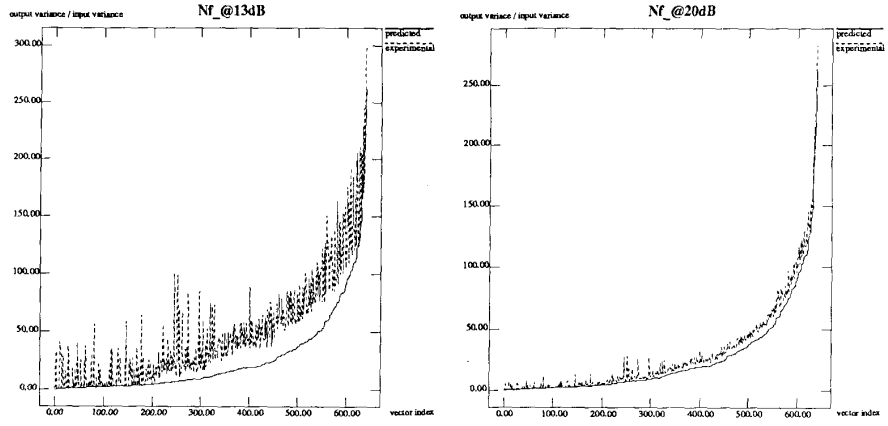

Fig. 13. Predicted and experimental noise factors.

a priori which classifier will have better performance at high SNR.

2) Results: Our experimental results bear out some of the above relationships. First, we examine the validity of linearizing the network for large SNR. Fig. 12 shows the relationship between the input and output noise variance for four vectors in the training set of a given classifier. The data has been plotted on linear and log-log scales to make the "linear" region stand out. As seen, typically the linearity assumption holds if the input noise variance does not exceed $\sigma_n^2 \approx 0.025$, which corresponds to an SNR of 13 dB in the simulations presented in Section III-B.

Secondly, we confirm that the expression obtained for the noise factor, (4), actually corresponds to the ratio of output/input noise variance for a classifier operating in the linear region. For each vector in the training set of the network with $S = 2$, $O = 5, H = 10$, we calculated the theoretical noise factor Nf_X, and estimated the output variance through simulation for two levels of input noise, SNR = 13 db, and 20 dB. The ordered results are shown in Fig. 13, which contrasts the experimental and predicted results for the entire training set. As expected, for even larger SNR the results match up closely. We note however that there is a large variation in the noise factors for vectors within a given training set, e.g., from 1 to 300. Indeed some of the vectors have particularly undesirable characteristics, in the sense that they would allow noise to propagate to the output and ultimately result in errors dominating the classifier's performance.

Finally, we calculate the P_e using (5). Table II shows both the experimental and theoretical P_e for coherent receivers at SNR = 13 dB. There are large numerical discrepancies between the predicted and experimental P_e although the qualitative behavior of the classifiers is preserved. Indeed the results confirm some of the trends suggested in Section III-B.

1) Increasing the number of hidden nodes results in better performance, but as the number of nodes becomes comparable to the number of input nodes, the improvement saturates. For example, a large improvement in performance results for $O = 7$ as H increases from 10, while for $H = 20$ and 30 the P_e is comparable. The same pattern holds for $O = 5$ and 9.
2) An increase of the observation window for a fixed large number of hidden nodes improves performance. For example compare lines 5, 8, and 11 or 4, 7, and

TABLE II
EXPERIMENTAL AND THEORETICAL P_e'S FOR COHERENT RECEIVERS

line number	S	O	H	$P_e \times 10^4$ @ 13 dB experimental	$P_e \times 10^4$ @ 13 dB predicted
1	1	5	20	19.06	6.54
	1	7	20	59.72	5.35
	1	9	20	159.74	44.34
4	2	5	10	85.27	5.44
	2	5	20	8.84	2.68
	2	5	30	6.61	0.50
7	2	7	10	209.34	68.65
	2	7	20	3.99	0.64
	2	7	30	4.16	0.80
10	2	9	10	60.88	5.34
	2	9	20	5.09	0.03
	2	9	30	< 1	0.49

10 in the table. Note however that when predicted and/or experimental P_e's are very small, predictions are unreliable; compare lines 6, 9, and 12.

3) Increasing sampling improves performance if there are a sufficient number of hidden nodes. Compare lines 1 and 5 or 2 and 8 in the table.

As seen in Table II, our estimates for P_e are somewhat optimistic. A closer look at the actual distribution of the signal reaching the output node reveals that it is not truly Gaussian, i.e., it is skewed, with a heavier tail than expected pointing away from the saturation region. Fig. 14, shows two examples of such noise distributions. The probability density of a random variable generated by passing Gaussian noise through a Sigmoid nonlinearity can be derived as shown in the Appendix B. Fig. 15 shows such distributions for noise of mean 1 and a range of standard deviations from 0.1 to 1.8. Clearly, such a random variable resembles a Gaussian for small variances (i.e., the large SNR case), but can have a radically different behavior as the noise variance is increased, or if the node is almost saturated. The probability distribution of the output noise is actually a weighted convolution of such distributions and inherits the skewing due to the saturating nonlinearity.

3) Discussion: A few comments about the above noise analysis are in order. Equation (6) shows that improved performance might be expected from a net whose hidden nodes are likely to be saturated, whose nonlinearities are very smooth (e.g., large T), and whose weights reduce the sums $\sum_{j \in \mathcal{L}_X} \alpha_j w_{ji}$ while still properly classifying the training set.[4] However one should be cautious in interpreting this statement. Indeed, the number of linear nodes, the temperature, and the weights are highly interrelated quantities resulting from or affecting the training algorithm. For instance an increase in the temperature, for the same training algorithm, does not necessarily improve the network's performance. In fact, we found experimentally that while increasing T from 1 to 2 results in a marginal increase of performance, further increases from 2 to 3 actually degrade the performance significantly (see Fig. 16). This may be explained heuristically by noting that convergence is slower when training at high temperatures

[4]Note that the terms in this summation are both positive and negative.

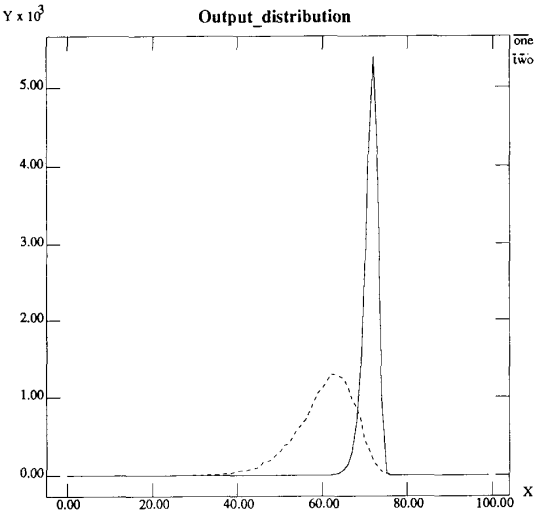

Fig. 14. Density of noise reaching the output node.

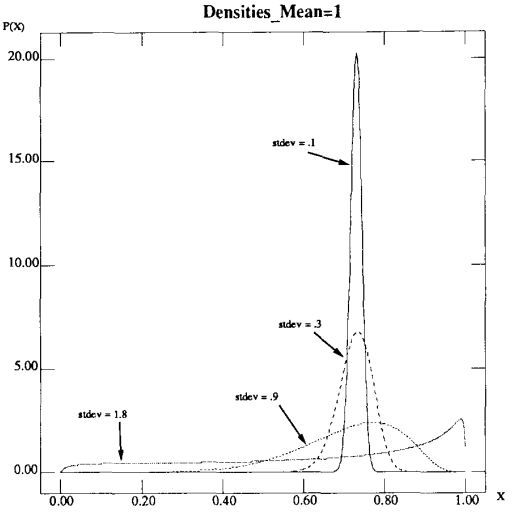

Fig. 15. Density of Gaussian noise passed through a Sigmoid.

and therefore results in weights for which hidden nodes are unlikely to be saturated.

We can suggest other ways of improving the noise performance of neural net classifiers. For example one could add the term $\sum_{i=1}^{I} \left\{ \sum_{j=1}^{H} \dot{\sigma}(m_j) \alpha_j w_{ji} \right\}^2$ to the cost function minimized by the training algorithm, in order to explicitly reduce noise propagation. Similar modifications have been proposed by Chauvin as means to obtain *optimal* use of hidden nodes [18]. However, our proposed modification, cannot be easily integrated into the backpropagation scheme. Indeed the noise characteristic of a network is a global property, in the sense that the effective amount of noise propagating through a given link depends directly on all of the weights in the network and the nominal state of each node [see (4)]. Since backpropagation relies on layer by layer updates, more direct optimization methods would be required.

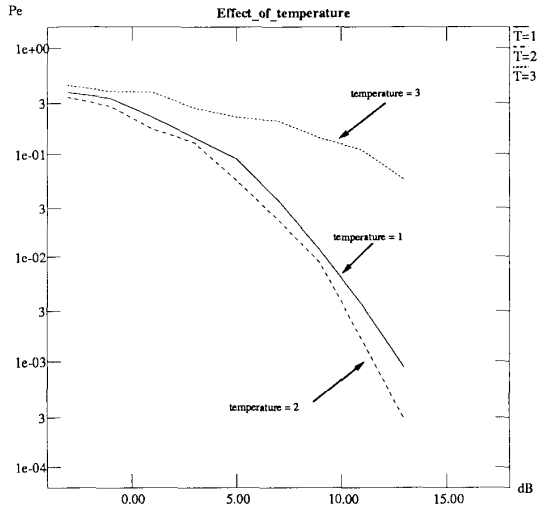

Fig. 16. Training at different temperatures: coherent 3RC network $O = 5$, $H = 20$, $S = 2$.

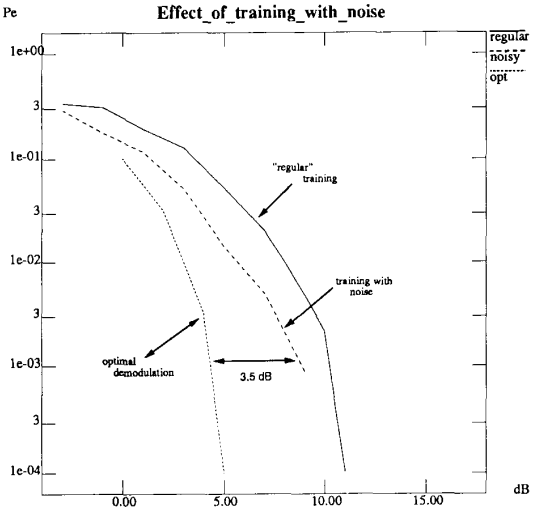

Fig. 17. Training at noise: coherent 3RC network $O = 9$, $H = 30$, $S = 2$.

Another intuitively pleasing way to improve the network's performance is to train with noise. Blanz and Gish [19] have indicated that neural net classifiers are impervious to training sets containing outliers.[5] This suggests that training with noise would still result in a viable network for vector classification while potentially improving network performance in noise. We tentatively verified this by using the following training scheme: first, we trained without noise until the network met the stopping criteria in Section III-b, then we trained with vectors having an SNR = 13 dB. The second phase lasted 10 passes through the entire training set, or approximately 100 000 iterations. As evidenced in Fig. 17 our best network's performance was improved by 1.5 db at $P_e = 10^{-3}$. We did not pursue this further even though, one can envisage many different training schemes which vary the temperatures as well as the noise level. Research on NN applications in nonlinear filtering for signal detection in non-Gaussian noise or image processing also exhibit the potential of training with noise [20], [21]. In particular it would be of interest to explore the possibility of using adaptive non-linearities, such as feed-forward networks, which could accommodate time varying channel or noise characteristics.

We have suggested several approaches to improve and possibly optimize the performance of NN classifiers in noise. Further study is required to determine which of these will pay off, and to what extent such approaches can improve the performance of our receiver.

V. Complexity Comparison

This section is devoted to comparing arithmetic complexity of conventional receivers to that of NN based receivers. This is particularly important since, as mentioned earlier, we can gain from using the NN paradigm if hardware requirements can be significantly reduced while performance is still reasonable.

[5] Training vectors whose distance is very large from the centroid of its class, when compared to the eigenvalues of the covariance matrix of that class.

Other methods of obtaining such a tradeoff also exist. In particular, reduced complexity Viterbi decoders have been proposed as a viable means to decrease demodulation complexity [6]. We begin with an overview of recent studies comparing NN classifiers with other statistical and classical classification methods.

A number of criteria can be used to compare trainable classifiers. These include implementation cost in terms of area and power, arithmetic and memory requirements, performance, training time, speed of decoding, and even training program complexity in terms of lines of code. In addition, it is important to examine classification tasks with different characteristics, such as the complexity of decision surfaces, presence of outliers in the training set and so forth. In their study of eight different types of classifiers, Lee and Lippmann noted that with rare exceptions similar error rates were obtained for the four applications considered [22]. They found however, that there was a wide variation in the required training period. For instance, on a serial computer, a k-nearest neighbor classifier trains 2–3 orders of magnitude faster than a multilayer feed-forward network. They also noted that NN classifiers made moderate usage of memory while classifying vectors quickly. Blanz and Gish's study also addresses the implementation cost of several classifiers in digital LSI. On the basis of their results, they conjectured that problems with large numbers of sample vectors and or categories can be more efficiently addressed by way of connectionist classifiers [19]. Blanz and Gish tested the susceptibility of classifiers to outliers in the training data set, and found that neural networks were particularly robust, although they point out that this is in part due to the iterative adaptation scheme. As mentioned in Section IV, this suggests that one could potentially train with noise and still get acceptable if not improved performance. Both studies indicate that there are many tradeoffs to be taken into account in selecting a classifier; this requires a careful study of the allowed hardware, restrictions on the training set, testing and adaptation time.

Other interesting results on the complexity of finite precision NN classifiers have been reported by Dembo, Siu, and Kailath [23]. After making some rather general assumptions, they show that perceptrons[6] require only $O(n \log n)$ bits of storage for parameters while a table lookup classifier requires $O(n^2)$. Haussler and Baum, tackle issues of complexity, sufficient training sample size and performance, in a somewhat more abstract setting; they establish a lower bound for the number of samples versus network size that is required for a NN to correctly classify examples drawn randomly from a given distribution [24], [25].

Due to the variety of possible complexity criteria available, the task of comparing our NN based receiver with conventional methods is nontrivial. We shall first restrict ourselves to a simplistic comparison of arithmetic complexity. A network with I inputs units, H hidden units and O output units requires $H \times (I + O)$ multiplies and additions per symbol interval. In a digital implementation, the Sigmoid would be evaluated by way of a table lookup, and $H+O$ lookups would be required per symbol interval. An analog implementation of the nonlinearity, is also conceivable, although this may degrade the precision of the nonlinearity. It is not clear at this point what level of accuracy is required in NN architectures [26], [19], [27]. Thus, our best performing NN with $I = 36, H = 30, O = 1$, requires about 1110 adds and multiplies, in addition to 31 Sigmoid evaluations per demodulated symbol.

For comparison, consider conventional CPM demodulation. The arithmetic complexity is proportional to the number of trellis states, $p \cdot M^{L-1}$. One can exploit the symmetry of the in phase and quadrature components to reduce the required number of matched filters to $2 \cdot M^L$, or 16 filters for binary 3RC. Suppose we use filters with 10 taps each,[7] then 160 multiplies and adds are required [5]. An additional overhead of 40×2 multiplies, 40×3 adds, and 80 trigonometric evaluations to be implemented by table lookup over the 5 possible phase states are required to form the inputs to the Viterbi processor (see Fig. 1) [5]. Viterbi decoders use efficient recursive algorithms to calculate the path metrics, however, they require large amounts of memory to keep track of the actual paths which will be used to ultimately make symbol decisions. At each stage, an add-compare-select function for each phase state is required, that is 40 adds and 40 compares. An additional, 20 compares are required to decode the symbol. An approximate total would give 320 adds, 220 multiplies, and 60 compares, in addition to 80 coarse trigonometric table lookups per stage. When compared to the figures obtained for the NN classifier, we conclude that NN numerical complexity approximately exceeds that of a conventional implementation by a factor of 3.

Finally, note that a digital implementation of a NN decoder would require the storage of 1110 weights in ROM versus the 160 filter coefficients needed in a conventional receiver. However, NN require virtually no RAM while a conventional receiver would need at least 220 RAM locations, to keep track of trellis paths. This is but a crude comparison of memory requirements, since the accuracy used to represent weights, filter coefficients and path metrics, has been disregarded, but it shows a tradeoff between ROM and RAM. This tradeoff is important, since RAM can take up 50% of the area of a custom chip of Viterbi decoding [28].

In spite of the above discussion, the NN architecture still has some advantages. The first is the regular and parallelizable nature of NN, which one might easily map to homogeneous architectures, such as systolic arrays, which are particularly suitable for VLSI implementations. Indeed the Viterbi algorithm is an inherently serial algorithm in that previous decoded symbols are required to decode the present and future ones. Parallel implementations of the Viterbi algorithm based on block partitioning of the data require synchronization periods, in which the initial state of the trellis path is estimated, before decoding can begin [7]. This is a fact the main reason for overlapping the partitioned blocks. The abovementioned synchronization period can potentially limit the extent to which the algorithm can be parallelized. The NN implementation studied has no feedback, thus by using more than one network in parallel, we can make the demodulation rate arbitrarily large. A second advantage lies in the speed of demodulation. Clearly, NN receivers would be faster since they require only a forward pass, while conventional decoders usually backtrack in order to obtain demodulated symbols. A third potential advantage, is to provide a form of nonlinear equalization by allowing online NN weight adaptation to the noise characteristics of the channel [20]. A fourth advantage, might arise from a favorable scaling of the cost of a NN receiver to an increase in the complexity of the modulation scheme. Indeed, it has been noted that as the problem size increase, i.e., number of features or classes, the NN architecture has an implementation cost advantage over other statistical and polynomial classifiers [19]. This point is more compelling when one notes that conventional methods scale as M^L, in both arithmetic and memory requirements.

Actually, from an implementation point of view, one could hardly expect a general purpose classifier such as a neural network, to compete with special purpose hardware, which exploits the particular structure of a problem. The implementation of Viterbi decoders in VLSI, is a subject of current research. Several impressive chips have been produced for decoding convolutional codes at very high data rates [28]. Research on NN architectures is still in its infancy. Important issues still need to be addressed such as how to maximally exploit the regularity of NN architectures, how to implement the required connection density, and how to provide some fault tolerance.

VI. Summary and Conclusions

We developed a class of neural net-based receivers for CPM schemes. From a functional point of view, these receivers are equivalent to the bank of matched filters and the Viterbi decoder found in conventional optimal ML receivers. Preliminary simulations, show that for binary 3RC with $h = 0.8$ the performance is within 3.5 dB of the optimal ML receivers at

[6] These are the simplest form of feed-forward networks, i.e., no hidden layers.

[7] We are assuming a reasonable sampling, of course this needs to be optimized.

$P_e = 10^{-3}$. We expect this margin to be partially reduced by optimizing both the architecture and training procedure. A comparison of arithmetic complexity for conventional and NN binary 3RC receivers shows that a traditional approach is still superior. However, in the light of the comparisons done by Blanz and Gish we believe these receivers will scale advantageously as the complexity of the CPM scheme increases [19]. The burden of training such networks will certainly decrease as new algorithms are reported to be orders of magnitude faster than back-propagation [29]. Finally if effective parallel architectures are developed for adaptation, a receiver could do online training, to accommodate specific channel characteristics.

An approach for analyzing the noise performance of multilayered feed-forward NN classifiers at large SNR has been proposed. We found that for large SNR, the ratio between the output and input noise variance can be analytically predicted and depends on the number of saturated nodes, the product of the input an output weights, and the temperature parameter of the nonlinearity. Our predictions of P_e only match the experimental data qualitatively, allowing us to find *a priori* which networks will have a better performance. Further work is needed to exploit these findings in developing effective training algorithms.

This particular application of the neural nets shows their capability beyond conventional associative memories where a fixed number of exemplars are learned. As demonstrated by the incoherent receiver, by training the net to classify a subset of the possible input/output pairs, one obtains a network capable of demodulating the entire signal space. In this sense, the net has captured that structure of the sampled received signal as well as the demodulation function.

Further work is required to study both the complexity and performance of the proposed NN approach for a more varied range of CPM schemes, under more realistic noise conditions. In particular, the possibility of adaptation to time-varying channels, and the use of preprocessing would be interesting. It may also be possible to extend the proposed analysis method using a more complex model based on the noise characterization derived in appendix.

Appendix A
Accuracy of the Probability of Error Estimator

In our experiments the probability of error of our classifier was estimated by simulating the receiver on a random sequence of inputs until 100 errors occurred. However, in light of the results of Section IV one might question whether this procedure is appropriate. Namely, we found that some of the vectors in the training set rarely result in errors while others have a relatively high probability of error. The discussion that follows addresses this issue in order to establish whether the experimental figures we obtained are valid.

First, we consider a single input vector X and the estimator of its probability of error $\hat{P}_{e_X}$ based on E_{Q_X} the number of errors that occurred during Q_X trials, $\hat{P}_{e_X} = \frac{E_{Q_X}}{Q_X}$. Assuming that the true probability of error is P_{e_X} we find that the estimator $\hat{P}_{e_X}$ has a binomial distribution with mean P_{e_X}, and variance $P_{e_X}(1 - P_{e_X})/Q_X$. It is thus unbiased and consistent.

By requiring that the standard deviation of the estimator be $\frac{1}{10}$th of the mean (i.e., an error of about 10%) we obtain,

$$0.1P_{e_X} \approx \sqrt{\frac{P_{e_X}(1 - P_{e_X})}{Q_X}} \approx \sqrt{\frac{P_{e_X}}{Q_X}} \quad \text{and}$$

$$Q_X \approx a\frac{100}{P_{e_X}}.$$

Thus, for each vector in the training set we require Q_X trails where Q_X depends on the actual probability of error of that vector.

The $\hat{P}_e$ of the classifier for the entire set of all possible input vectors is estimated by the average $\hat{P}_{e_X}$ of all vectors. This guarantees that the overall standard deviation of the estimated probability of error is $\frac{1}{10}$th of its mean. However, this procedure requires lengthy simulations to obtain independent estimates of $\hat{P}_{e_X}$ for each vector belonging to the training set. Moreover some of the vectors rarely result in errors, so they need to be simulated extensively to find accurate predictions.

In order to check that our original approach was valid, and that the discrepancies in the theoretical and predicted probabilities of error were not the result of our testing procedure we conducted some limited simulations to check our estimates for P_e in the more systematic manner we have described. We found that our original figures were quite reliable.

Appendix B
PDF of Gaussian Noise Passed Through a Sigmoid

The Sigmoid function is written as

$$y = f(x) = \frac{1}{1 + e^{-x}}$$

can be inverted to obtain

$$x = f^{-1}(y) = \log\left\{\frac{1}{1 - y}\right\}.$$

If we let x be Gaussian variable with mean m, and variance σ,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-m)^2}{\sigma^2}}$$

we can write the cumulative distribution function for the output variable y as

$$P(y) = \Pr\{Y \leq y\} = \int_{-\infty}^{f^{-1}(y)} p(x)\,dx.$$

Finally differentiating $P(y)$ we obtain the density of the output variable,

$$p(y) = \frac{1}{\sqrt{2\pi\sigma}} \frac{1}{y(1 - y)} e^{-\frac{1}{\sigma^2}\left\{\log\frac{y}{1-y} - m\right\}^2}.$$

The probability density will be markedly different, as the input mean and variance are varied (see Fig. 15).

REFERENCES

[1] A. Zahkor and E. Bedrosian, "Advanced modulation techniques for amber (am broadcast emergency relay)," A rand note n-2393-arpa, Rand's Eng. Appl. Sci. Dep., Santa Monica, CA, 1985.

[2] J. Anderson, C.E. Sundberg, T. Aulin, and N. Rydbeck, "Power-bandwidth performance of smoothed phase modulation codes," *IEEE Trans. Commun.,* vol. COM-29, 1981.

[3] J.D. Provence, "Neural network implementation for maiximum-likelihood sequence estimation of binary signals," in *Proc. IEEE ICNN,* 1987, pp. 703–714.

[4] G. Zeng, D. Hush, and N. Ahmed, "An application of neural nets in decoding and error correcting codes," in *ISCAS,* 1989.

[5] J.B. Anderson, R. Aulin, and C.E. Sundberg, *Digital Phase Modulation.* New York: Plenum, 1986.

[6] T. Aulin, A. Svensson, and C.E. Sundberg, "A class of reduced-complexity viterbi detectors for partial response continuous phase modulation," *IEEE Trans. Commun.,* vol. COM-32, 1984.

[7] P. Black and T. Meng, "A hardware efficient parallel Viterbi algorithm," in *ICASSP,* 1990.

[8] J. Heller and I. Jacobs, "Viterbi decoding for satellite and space communication," *IEEE Trans. Commun.,* vol. COM-19, 1971.

[9] K.-H. Tsou and J. Dunham, "Sliding block decoding of convolutional codes," *IEEE Trans. Commun.,* vol. COM-29, 1981.

[10] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *IEEE Proc. ICNN,* 1988, p. III 11–13.

[11] B. Irie and S. Miyake, "Capabilities of three-layered perceptrons," in *IEEE Proc. ICNN,* 1988, p. I 641–648.

[12] K. Funahasi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks,* vol. 2, pp. 183–192, 1989.

[13] A. Lapedes and R. Farber, "How neural nets work?" Preprint la-ur-88-418, Los Alamos Nat. Lab., Los Alamos, NM, 1988.

[14] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by backpropagating errors," *Nature,* vol. 323, pp. 533–536, 1986.

[15] D.E. Rumelhart and J. McClelland, *Parallel Distributed Processing Vol. 1* Cambridge, MA: M.I.T. Press, 1986.

[16] G. Mirchandani and W. Cao, "On hidden nodes for neural nets," *IEEE Trans. Circuits Syst.,* vol. 36, May 1989.

[17] S.Y. Kung and J.N. Hwang, "An algebraic projection analysis for optimal hidden units size and learning rates in back-propagation learning," in *IEEE Proc. ICNN,* 1988, pp. I 363–370.

[18] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units," in D. Touretzky, editor, Neural Network Information Processing 1, D. Touretzky, Ed. Morgan Kaufmann, 1988, pp. 519–526.

[19] S. Gish and W. Blanz, "Comparing a connectionist trainable classifier with classical statistical decision analysis methods," Res. Rep., IBM, Almaden Res. Cen., San Jose, CA, 1989.

[20] R. Lippmann and P. Beckman, "Adaptive neural net preprocessing for signal detection in non-gaussian noise," in Neural Network Information Processing 1, D. Touretzky, Ed. Morgan Kaufmann, 1988, pp. 124–132.

[21] K. Arakawa and H. Harashima, "A nonlinear digital filter using multi-layered neural networks," in *IEEE Int. Conf. Commun.,* vol. 2, pp. 306A.2, 1990.

[22] R. Lippmann and Y. Lee, "Practical characteristics of neural network and conventional pattern classifiers on artificial an speech problems," in *Neural Network Information Processing 2,* D. Touretzky, Ed. Morgan Kaufmann, 1990, pp. 168–177.

[23] A. Dembo, K-Y. Siu, and T. Kailath, "Complexity of finite precision neural network classifiers," in *Neural Network Information Processing 2,* D. Touretzky, Ed. Morgan Kaufmann, 1990, pp. 124–132.

[24] D. Haussler, "Generalizing the pac model for neural net and other learning applications," Ucsc-crl-89-30, Comput. Res. Lab., Univ. California, Santa Cruz, CA, 95060, 1989.

[25] E. Baum and D. Haussler, " What size net gives valid generalization?" in *Neural Network Information Processing 1,* D. Touretzky, Ed. Morgan Kaufmann, 1988.

[26] T. Baker and D. Hammerstrom, "Characterization of artificial neural network algorithms," in *ISCAS,* 1989.

[27] M. Takeda and J.W. Goodman, "Neural networks for computation: Number representations and programming complexity," *Appl. Opt.,* vol. 25, no. 18, 1986.

[28] R. Kerr, H. Dehesh, A. Bar-David, and D. Werner, "A 25 mhzk viterbi fec codec," in *Proc. IEEE Custom Integrat. Circuits Conf.,* pp. 13.6.1, 1990.

[29] R. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks: Application pattern recognition," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, no. 12, Dec. 1989, in review.

[30] R.P. Lippmann, "Pattern classification using neural nets," *IEEE Commun. Mag.,* vol. 27, pp. 47–64, Nov. 1989.

[31] E.A. Lee and D.G. Messerschmitt, *Digital Communication.* New York: Kluwer-Academic, 1988.

Gustavo de Veciana (S'89) was born in Uruguay in 1966. He received the B.S. and M.S. degrees in electrical engineering from University of California at Berkeley in 1988 and 1990. He is currently pursuing the Ph.D. degree at Berkeley.

His research interests are in communications, neural networks, queueing, and traffic models.

Avideh Zakhor (M'87) for a photograph and biography, see this issue, p. 1387.